SU3

Data Migration

Exam Guidance

- You may use these slides to guide your preparation FROM THE TEXTBOOK
- The code examples are not required for the exam, but try to be at least able to read the code

Introduction

- The aim of the study unit is to increase your awareness of the need and functionality of NOSQL
- To understand the big ideas of data migration
- To increase your understanding of different types of data stores

Need for NoSql and data migration

- Sources of data 2.5 *10^{18} (EB) per day!
- Different formats make the ETL cannot keep up!
- Different devices and methodologies!!

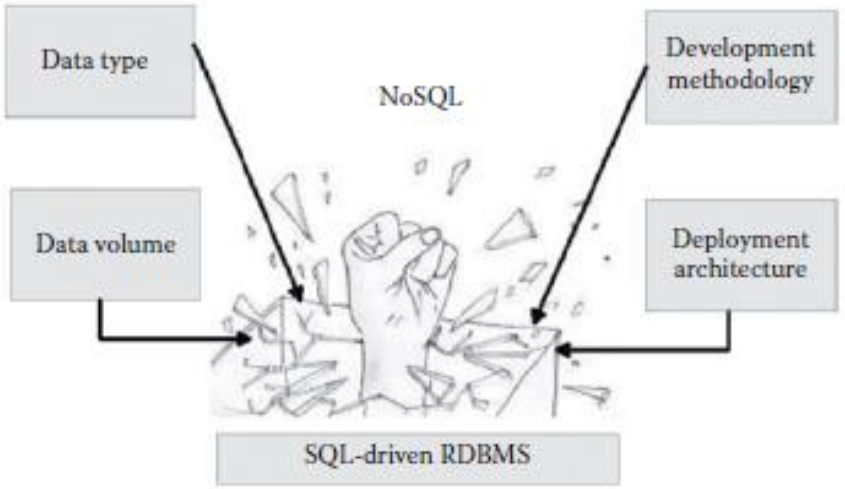


TABLE 3.1

Characteristics of "Big Data"

Type of Data	Volume of Data
• Unstructured • Semistructured • Polymorphic	• Petabytes • Millions of records • Trillions of queries per second

Horizontal scaling out vs Vertical Scaling up

- In horizontal scaling, when there is an increase in application performance expectations and load, additional servers are added.
- This leads to running multiple instances of the application in parallel on each of these additional servers
- Vertical scaling on the other hand refers to upgrading the existing server with powerful hardware. This is often called “scaling up” and often involves downtime.
- RDBMS is heavily dependent on SQL for its ETL operations. SQL has huge performance issues when dealing with distributed applications that are “scaled out” as it involves intelligent partitioning of the database, writing optimized set of SQL Join operations, Sub-queries, and Stored Procedures across multiple parallel database instances of the application deployed on a set of servers.

Need for NoSql and data migration

Problems (explain what is meant by these in context of the specific example):

- Complex SQL queries
- Highly normalized schema not aligned to multiple data types
- Very poor horizontal scalability
- Unpredictable peak loads
- Walmart: Too much delay in producing real-time data aggregation for data analytics
- Job posting articles have large metadata attached to them, for example, abstracts, information about publisher, and biographies.
- Job Posting: Metadata is highly unstructured.
- Large quantities of image and video data to be stored, retrieved, and updated without read and write latency.
- EBAY : Not scalable to handle billions of reads and writes each day to be processed at blistering speeds.
- ACID transactional constraints of traditional RDBMS.
- Not efficient to do rapid analysis on a broad assortment of structured and unstructured data captured real time, while customer is online.

Need for NoSql and data migration

Positives after migration

- Mongo:
  - Dynamic schema creation using JSON
  - Social Network Analysis using light-weight map-reduce algorithms
- Neo4j:
  - Graph databases can quickly query customers' past purchases as well as instantly capture new interests shown in his/her current online visit.
  - Matching of historical and current session data facilitates real-time relevant recommendations for the customer.
- Amazon Dynamo
  - Eliminates tons of attributes. Instead, use single document per user/article.
  - Simple scalability and high availability.

- Cassandra:
  - Robust support for clusters spanning multiple data centers
  - Encourages data redundancy through data replication
  - High availability and no scope of single-point failure.
- For Exam: Make a list of typical problems that motivated these organisations to migrates

Data Migration

- Typical steps!

- *Step 1*: Connect Mongify to SQL and NoSQL databases. Here, they are MySQL and MongoDB, respectively.
- *Step 2*: Generate the translation file. Translation file is a ruby file that provides mapping instructions to Mongify.
- *Step 3*: Execute commands to convert the SQL rows to MongoDB documents.

Always consider

- General SQL import
- Importing specific rows or columns
- Incremental imports

Mongo DB JSON

TABLE 3.5

Snapshot of MongoDB Document after Migration

```
{
  "_id" : ObjectId("56cdc0874e05e51a9c000001"),
  "firstname" : "Lionel",
  "lastname" : "White",
  "FavoriteFood" : [
                      {
                        "food" : "Bacon"
                      }
                  ]
}
```

```
> db.MaryPerson.find({})
{ "_id" : ObjectId("56fbdf094e05e51cc9000001"), "firstname" : "Mary",
"lastname" : "Lambert", "Mary" : [ { "food" : "Fries" }, { "food" : "N
achos" } ] }
>
```

Sqoop into Hadoop

- The motivation for data migration from MySQL to a Hadoop cluster is two-fold. The data present in the MySQL database are too large to be processed through an SQL query in a reasonable amount of time, or the data are infrequently accessed and it is better to store these large data in Hadoop, which is optimized for dealing with storing large datasets like usage logs, error logs, etc. In this section, we will be considering data migration from a MySQL database to a Hadoop cluster;
- Data migration from MySQL to Hadoop can be done manually by exporting SQL data into a format like CSV, and importing the generated CSV into HDFS using any simple scripting language like Perl, Python, or Ruby.

SQL to Neo4J: Data Migration Using Neo4J

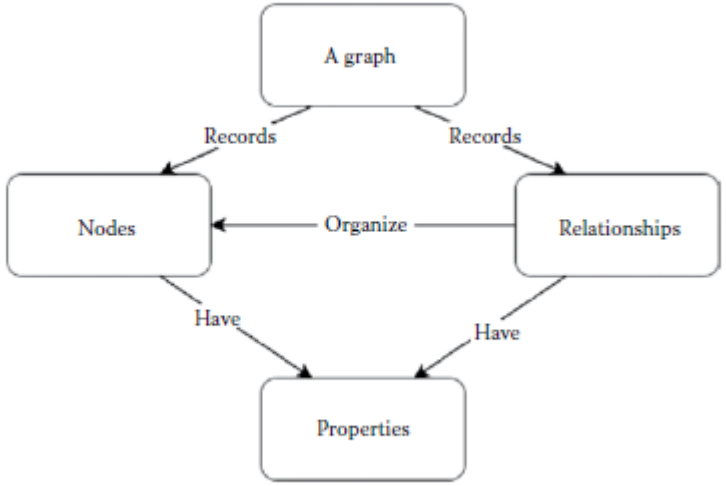
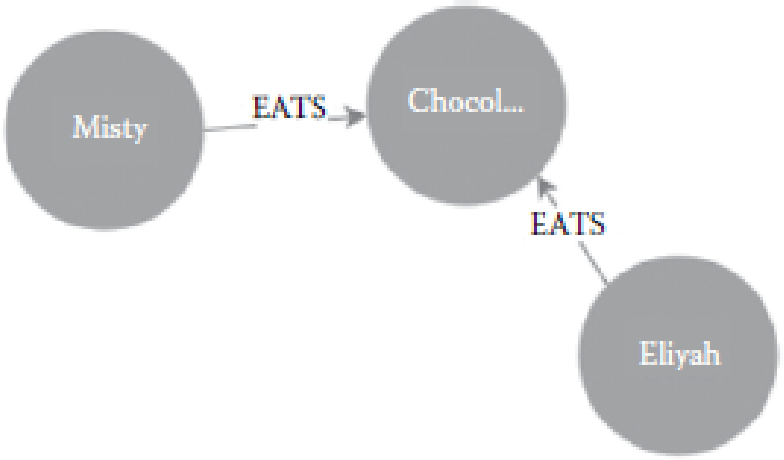


FIGURE 3.6

TABLE 3.19

Adding Relationship "EATS" Using MERGE

```
LOAD CSV WITH HEADERS FROM "file:///tmp/person.csv" AS csvLine
MATCH (desserts:Desserts { did: TOINT(csvLine.D_ID)})
MATCH (person:Person { did: TOINT(csvLine.D_ID)})
MERGE (person)-[:EATS]->(dessert)
return person,dessert;
```



Step 1: Exporting SQL table to CSV with headers
Step 2: Import/load CSV to Neo4j
Step 3: Add relationships to the nodes

Google Cloud Dataflow for Pipelining Data across Storage Systems

- Using Cloud Dataflow, we can define data sources and sinks to enable pipelining of data from one source to a sink, which can be of different structures.
- Due to lack of a standardized approach to data migration, developers often tend to use custom in-house scripts for handling specific data migration jobs, where the script is configured for only a certain type of data and for migration between fixed set of systems. However, attempts at one tool for all databases have been made, and one such attempt is exemplified by an independently developed tool titled “SQLToNoSQLImporter”